Supporting ONME with a method engineering framework

O. Pastor, J.C.Molina, B. Henderson-Sellers
Universidad Politecnica de Valencia; CARE Technologies; University of Technology, Sydney
opastor@dsic.upv.es; jcmolina@care-t.com; brian@it.uts.edu.au

Abstract. ONME is an environment for automated software engineering, focussed on the modelling activity. To make it more useful for practical software development, we here augment it with a full process derived from the metamodel contained in the OPEN process framework. Then, using the method fragments in the OPEN repository, we use method engineering techniques to generate the ONME approach in this augmented format. We also discover two work products not currently described in the OPEN repository that are necessary for complete support of the ONME approach.

Keywords. Automated software engineering, method engineering, situational method engineering, Model Driven Architecture

1. Introduction

ONME (OlivaNova Model Execution) is an environment for automated software engineering and is an implementation of OO-Method [17], itself based on the formal conceptual modelling work of OASIS [20]. With ONME, a Conceptual Model is translated into a final software product in an automated way by applying a model compiling strategy, where the mappings between conceptual primitives (problem space) and their corresponding software representations (solution space) are properly defined.

The practical use of the conceptual modelling approach associated with ONME has revealed the need for solid process support in order to apply ONME correctly in industrial settings. ONME is intended to be used by different software providers, who must build the corresponding object-oriented conceptual schemata that constitute the input of ONME's MDA[1]-based engine that takes the PIM (Platform Independent Model) and produces the final software product (essentially a PSM or Platform Specific Model) [15]. ONME provides code generation capability but offers no advice to users on how to design the conceptual models, elicit requirements, undertake project management, test the quality of the output system etc. All these must be provided by embedding the ONME MDA-based approach into a more appropriate methodological framework, thus providing full lifecycle support.

In this paper, we enhance the ONME approach by adding one such complete process environment. This is provided from the OPEN[2] approach [7], which uses a method engineering framework. This framework consists of a metamodel and a repository of method fragments (as described in detail in Section 2). Fragments are selected from the repository in a standard method engineering (ME) approach (e.g. [3,24]), itself described in more detail in Section 3.

The description of ONME in Section 4 is a prelude to our analysis of how we might integrate the ONME conceptual models with the OPEN framework in such a way that we can create process and methodological support for users of the ONME environment. We do this in Section 5 by analyzing the tasks and techniques suggested in ONME and identifying the method fragments in OPEN that support these. As a final result, this work provides two main practical contributions:

[1] Model-Driven Architecture
[2] Object-oriented Process, Environment and Notation

- The "open" character of the OPEN approach is assessed, by analyzing and putting into practice its capacity to adapt to the particular aspects of the ONME approach
- A precise software process support for successful use of an advanced, MDA-based method is defined

In summary, in this paper we develop the theory and rationale for the merger of an MDA tool into an existing method engineering methodological approach We do not attempt to provide any empirical results here, but intend to do so in a further paper following planned empirical studies

2. OPEN

OPEN (Object-oriented Process, Environment and Notation) [6,7,11] is an established approach for developing software, primarily that with an object-oriented implementation It comprises a metamodel that defines all the methodology[3] elements at a high level of abstraction plus a repository that contains instances of those metalevels concepts (Figure 1) It is thus highly compatible with the ideas of method engineering and process construction as described below

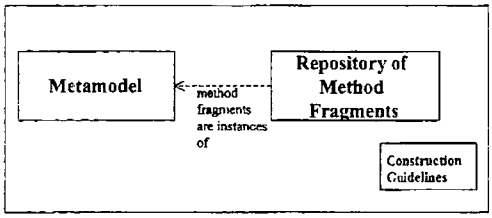

Figure 1 A metamodel provides a set of definitions from which can be created a large number of instances. When modelling a methodology, the repository stores method fragments.

From this framework, OPEN-compliant processes can be instantiated to be used in actual software projects within an organization In other words, the process engineer and/or project manager has to configure OPEN, creating an instance of OPEN that is suitable for use on a specific project. A company-customized OPEN version is then "owned" by the organization, becoming their own internal standard, while retaining compatibility with the global OPEN user community (http://www open org au)

While the OPEN metamodel contains numerous metalevel concepts, there are only five top-level concepts:

- Work Product: "A Work Product is anything of value that is produced during the development process" [6, p44] Work Products are the result of producers (people) executing Work Units and are used either as input to other Work Units or delivered to a client Pragmatically, they also include externally-supplied (e g by user) pre-existing artifacts used as inputs to Work Units
- Producer: "A Producer is responsible for creating, evaluating, iterating and maintaining Work Products" [6, p50]
- Work Unit: A Work Unit is defined as a functionally cohesive operation that is performed by a Producer There are three major classes of Work Unit: Activity, Task and Technique Activities and Tasks describe "what" is to be done, the former at a largescale granularity (e g that of a team), the latter at a more detailed level (e g that of an individual developer) In contrast, a Technique describes "how" the Task is undertaken.

[3] We use a definition in which the term methodology encompasses both process and product [22]

- Language: A Language is defined as a medium for documenting a Work Product
- Stage: A Stage is defined as an identified and managed duration within the process or a point in time at which some achievement is recognized

Each of these metaclasses has many subclasses in the detailed metamodel (see Appendix G of [6]). From each of the leaves in the metamodel hierarchy can be generated one or more method fragments, which are, in turn, documented and stored in the OPF[4] repository (Figure 1) This is an important part of OPEN This repository provides an almost complete set of components and can be readily extended to support changes in technology (as is undertaken here) It currently contains about 30 predefined instances of Activity, 160 instances of Task and 200 instances of Techniques.

Method fragments from the OPEN repository are linked together with the advice contained in a set of deontic matrices The values in these matrices are set to one of five levels of possibility: mandatory (M), recommended (R), optional (O), discouraged (D) and forbidden (F) – although a smaller number of values can also be applied e g binary, particularly in the early stages of process adoption

The values in this matrix will vary depending upon a number of factors such as project size, organizational culture, domain of the application to be developed, and the skills and preferences of the development team For example, based on the specific values of these characteristics, the possibility value for using the OPEN Task: Evaluate quality to help fulfil the Activity: Verification and Validation (V&V) might be assessed as being "Recommended" (one of the five prescribed values) Tool support for completing these matrices is currently under development [14]

Activities and Tasks state what is to be done, but make no suggestion as to how these are accomplished Techniques to do this are fully documented also in the OPEN repository [11], which brings together all the well-tested OO (and other useful) techniques that have been used worldwide for the past decades Users of the OPEN methodological approach utilize a second deontic matrix to help them choose the Technique most likely to be successful for their particular Task, bearing in mind the skills and preferences of the people directly involved For example, there are several Techniques for finding objects Some OO software developers start with a textual analysis, some use simulation, some use CRC (Class Responsibility Collaborator) cards [1] and yet others prefer a use case-driven method New Techniques can be readily added to the matrix as an extra line

The combination of Tasks and Techniques creates one or more work products which, in turn, are consumed by other Tasks These Work Products include plans, reports and graphical models, the last usually documented by use of the Unified Modeling Language, UML (see [9])

These matrices are part of OPEN's use of method engineering (see next section) to construct the whole methodology from the repository's method fragments The repository also contains a number of guidelines [6] for helping organizations to adopt OPEN as a standard for their information system development projects

3. Situational method engineering

Method engineering [3,26], or more appropriately situational method engineering or SME [25], recognizes the fact, often ignored, that seeking to develop an all-encompassing methodology appropriate for all situations is foolish [4] Rather, SME seeks to model methodological processes and products by isolating conceptual method chunks or method

[4] OPEN Process Framework

fragments [22] From these method fragments a specialized methodology can be constructed that is appropriate for each and every situation When an organization develops software in a single domain, that constructed methodology may suffice for all developments; for other commercial developers, the constructed methodology may need to be project-specific The methodologist publishes the method fragments, usually contained in a methodbase or method repository together with some construction guidelines [2,13,21]; and the in-house method engineer uses these construction guidelines to create the organization-specific or project-specific methodology within the organizational context and constraints (Figure 2)
Ideally, the elements in an SME repository should be compliant with (in fact generated from) a set of concepts described by a metamodel [10] - as we have seen in Section 2 for the OPEN methodological approach The concepts most relevant to the creation of ONME (as will be discussed in Section 4) are (i) Task, (ii) Technique and (iii) Work Product Each of these metaclasses can be instantiated to create numerous instances of Task, Technique and Work Product respectively, all of which are stored in the OPEN repository (Figure 1)

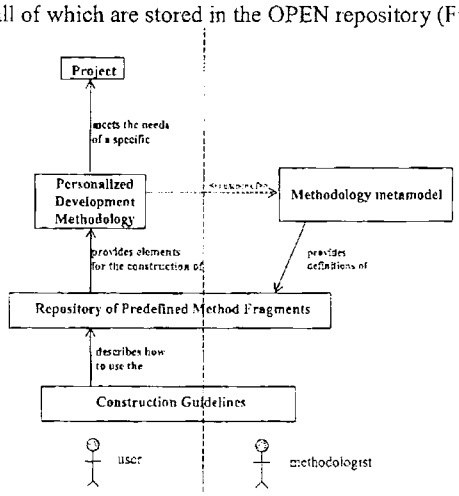

Figure 2 (right hand side) The methodologist is responsible for the process metamodel, creating a significant number (but not all) of the process components in the repository and the guidelines for construction (left hand side) The user/method engineer uses the construction guidelines and the contents of the repository (which they are at liberty to add to) to create a "personalized development process" attuned to a specific project or context.

4. ONME

In this section, we introduce OlivaNova Model Execution (ONME), an implementation of OO-Method, an object-oriented methodology for the development of software applications It was created by CARE Technologies, a spin-off from the Technical University of Valencia, in the context of a fruitful cooperation between academia and industry
A precise software process is a main component for any software development method to be considered sound This has been experienced by CARE Technologies practitioners when acting as software providers using their own technology In the first software projects developed by CARE itself, the need to have such a precise software process was seen as a strong facilitator for achieving the goal of obtaining a resulting software product that is both correct and complete, especially considering the diverse set of situations faced when using the

methodology in different environments (e g variations in profiles of technical staff, in local technological culture of different companies)
In some sense, this is a well-known although typically unsolved problem. Some years ago Entity-Relationships Diagrams were introduced as a way for doing correct data modelling work; however, no details were normally given about how to deal with a very basic issue: how to detect relevant system entities and their relationships
Even nowadays, in order to provide a rigorous software production process, it is not enough to list the number of UML diagrams to be constructed during process execution. Often, well-known methods require any type of Use Case Model, Class Diagram, Interaction Diagrams, . and so on The problem starts when this has to be done in practice for concrete, complex systems, when no precise guidelines are provided for the serious accomplishment of such a software production process through the use of the corresponding set of tasks, techniques and work products. This is a major problem for many methods in order for them to be successful in industrial practice
The problem to be solved in this context is twofold:

- how to select the appropriate software process framework,
- how to put it into practice, through the proper component definitions and the corresponding framework update when required

We discuss now these two aspects Why have we selected OPEN? As noted earlier, OPEN is a very convenient approach to characterize precisely such a kind of software process, by defining tasks, techniques and work products associated with a given method Due to its precise metamodelling approach and its open philosophy, it is really feasible to adapt and extend it to a particular approach (as ONME) by properly modifying its basic repository of method fragments. This provides a practical way of putting it into practice, answering the second question above Accordingly, the problem solved in this paper is how to provide a precise software process for ONME based on OPEN An added-value of this work is the successful experience acquired in using a software process framework in an advanced, MDA-based software development environment provided by ONME [19]
To develop these ideas, we first introduce the most relevant characteristics of ONME in this section. Next, the phases and models needed to accomplish an ONME software development process are presented Finally, in Section 5, the proposed ONME software process is characterized as an instance of OPEN, by specifying OPEN method fragments for each corresponding ONME task, technique and work product

4.1 ONME fundamentals

The idea of generating code from models has been implemented in a variety of ways with varying success. Typically, the process is based on the use of some UML-compliant diagrams [16]. The amount of code automatically produced from those diagrams is relatively small compared to that of a fully functional application. Consequently, hand-coding is generally still the most significant phase in the process of applications development. Such implementations can be catalogued as MDCG (Model-Driven Code Generation) tools, because models are used to assist developers in the task of programming of software applications In other words, models can be seen as guidelines but are never considered to be true 'programming artefacts'.
ONME's proposal goes one step further It constitutes a real Model-Based Code Generation (MBCG) set of tools. Unlike Model-Driven tools, in Model-Based Code Generation tools, models play a central role in the development of applications and can be regarded as true programming artefacts While it is true that MBCG tools have been successfully applied in certain domains like real-time and embedded systems, the goal of OlivaNova Model Execution is to provide an MBCG solution for information systems in general.

The technical background is simple yet powerful The relevant conceptual primitives required to specify an information system are defined precisely. A set of UML-compliant diagrams that represent them is provided, conveniently complemented with the required textual information. This provides a kernel of UML modelling constructs, where the huge variety of concepts provided by UML is reduced to a precise subset of conceptual primitives. Every conceptual primitive has its corresponding software representation counterpart. The implementation of this set of mappings between conceptual primitives and their corresponding software representations is the core of what in ONME terms is called a Conceptual Model Compiler

4.2 OO-Method fundamentals

The OO-Method [17], upon which ONME is built, focusses on a clear separation of the Problem Space (the 'what') from the Solution Space (the 'how') The definition of a problem (the abstract description of an application, represented in the corresponding Conceptual Schema), can occur independently from any concrete reification (concrete implementation of a solution).

The formalism behind the OO-Method is OASIS [18,20], an object-oriented formal specification language for information systems This formal framework provides a sound characterisation for the conceptual constructs required to specify an information system Its two main components are the Conceptual Model and the Execution Model.

The Conceptual Model is divided into four complementary views (Figure 3): the object view, the dynamic view, the functional view and a fourth view to specify user interfaces These four views allow all the functional aspects of an application to be described in an abstract manner by means of a set of conceptual constructs (also called conceptual primitives or conceptual patterns) that have clear, precise semantics. Most of these conceptual patterns have a UML-based graphical syntax, which hides the complexity and the formalism of the underlying OASIS specification

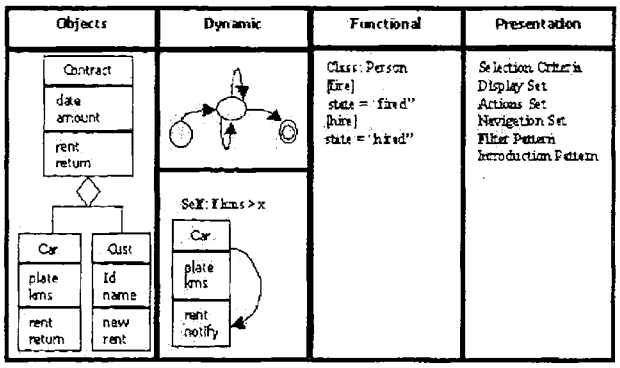

Figure 3 OO-Method's four viewpoints The Conceptual Model specification is structured in these four different but complementary system views: object (static architecture), dynamic, functional and presentation (user interaction).

The Execution Model defines the behaviour of objects belonging to the specified system (Conceptual Schema) at execution time The rules governing the conversion of the Conceptual Schema into its corresponding software representation, fully dependent on the target technological platform, are defined in the context of the Execution Model. The Model Compiler implements the set of mappings that relate conceptual patterns to software

representations, under the assumption that any programming decision has a conceptual counterpart that can be obtained and properly represented at a higher level of abstraction (the Conceptual Schema) ONME currently implements the conceptual model compiling process for a 3-tier software architecture, because this is a well-known, widely-extended strategy in industrial environments Nevertheless, this shouldn't be seen as a limitation, because the proposed transformation model-based approach can be applied to any software architecture, since every conceptual pattern has its corresponding software representation in any selected target software development environment

5. ONME Software Process as an instance of OPEN

The application of OPEN to characterize the ONME Software Process introduces some interesting aspects Since OO-Method provides a Model-Based Code Generation approach, two main phases can be distinguished:

1 The Conceptual Model construction (at the problem space level)
2 The Implementation (representation of the Conceptual Model at the solution space level)

The first important aspect is that the implementation work unit is obtained by the model compiler, as the result of an automated translation process. In consequence, the main engineering activities are centred on obtaining the Conceptual Model The question to be answered is, "What are the relevant tasks, techniques and work units required to build the Conceptual Model?"

5.1 Tasks charaterizing ONME

As the OO-Method expressiveness is built upon four models, these models provide the key to specifying the corresponding four ONME tasks:

1 Specification of the Class Model
2 Specification of the Dynamic Model
3 Specification of the Functional Model
4 Specification of the Presentation Model

In the first ONME task, the system has to be represented as a network of interacting objects The objective is to obtain a specification of the system class architecture, where all the static aspects of the main components of this object society must be precisely defined, including all classes and valid relationships between classes

In the second ONME task, the class model obtained in the first task is complemented with the representation of the dynamic system view, where objects are seen as processes whose lives can be formally represented as sequences of valid states, and where object intercommunication mechanisms are refined in terms of global transactions and trigger relationships

After having declared the system class architecture and its dynamic aspects, in the third ONME task these static and dynamic views of the object-oriented system are completed with the functional view, which states the definition of every class service in terms of what changes of state can be produced in the object acting as service provider

The pre-existing OPEN Tasks that can be mapped to these ONME tasks are just two: Identify CIRTs (viz classes, instances, roles and types); and Construct the Object Model. In OPEN, Tasks are at a slightly higher level of granularity than ONME tasks - the detailed level is specified by a number of OPEN Techniques e g for identifying association and aggregations relationships between classes, identifying specialization hierarchies, identifying agent relationships, declaring the set of static (attributes and integrity constraints) and dynamic

(services) class-scope properties, identifying valid states for class objects, specifying inter-class services (whose atomic actions belong to more than one class), capturing services activated in an automated way when a given condition holds and specifying the corresponding precondition and postconditions for every class service.
In the final ONME task (Specification of the Presentation Model), the important aspects of user interaction are taken into account. The task goal here is to specify the world view of any system user as a relevant part of the system specification. In this case, there is a specific OPEN Task that can be mapped to this ONME task: Design User Interface. As before, this then uses Techniques at a more detailed level to effect the UI design. These techniques are now explored in more detail in the following section.

5.2 ONME techniques

The OPEN repository collects an important number of techniques, but not all of them are needed in the context of ONME. The goal of this section is to identify which OPEN techniques are really relevant for defining a precise Software Process for ONME. The added value of this work is that it can be adapted to any other software production method. Furthermore, and as interesting further work, this provides in the context of MDA a challenging framework for comparing different MDA-based proposals, analysing those techniques, activities and work products required by them.

5.2.1 For specification of the class model. For specifying the Class Model, ONME suggests 1) identification of the relevant system classes (defining attributes, services and integrity constraints) and 2) identification of class relationships, distinguishing those of association/aggregation, specialization (permanent or temporal) and actor (which classes are allowed to activate which services).
In OPEN, there are a large number of pre-existing Techniques used to support the creation of the class model. These are readily mapped to (and can therefore be used to instantiate) ONME Techniques as follows. For identifying the relevant system classes, the reification of domain concepts in the form of classes is the main approach. This is supported, *inter alia*, by the following named OPEN Techniques: abstract class identification, class naming, collaborations analysis, creation charts, design templates, responsibility identification and stereotyping. Additionally, verbs within a class scope are an indication of class services, and nouns within a class scope can potentially refer to class attributes. Relationship-focussed techniques in OPEN include generalization and inheritance identification, interaction modelling, relationship modelling and service identification.
Other complementary techniques are the representation of nouns as classes (i.e. lexical analysis), representing relevant actions as class services (distinguishing between service server class and service client class), the identification of class taxonomies, the detection of "part-of" relationships, and the identification of binary associations (including the definition of their properties (multiplicity, etc.) - all subsumed in one or more of the existing OPEN Techniques.

5.2.2 For specification of the dynamic model. For specifying the Dynamic Model, ONME defines 1) validation of object lives, i.e. ensuring we have valid sequences of class services, 2) global transactions, i.e. those services in whose definition services of different classes are involved and 3) trigger relationships, to specify services that will be activated in an automated way when a given condition holds.
In OPEN, there are a number of pre-existing Techniques used to support the creation of the dynamic model. These are readily mapped to (and can therefore be used to instantiate) ONME

Techniques as follows For validating class object lives, identifying object traces composed of valid sequences of class service occurrences is required (e g OPEN Techniques of Object life cycle histories) For discovering global transactions, the required technique is to identifying scenarios for transactions seen as molecular execution units whose atomic components are services of different classes Here, OPEN Techniques found useful include Scenario development For detecting trigger relationships, performing an ONME agent service analysis for identifying those system actions that have no concrete actor is the technique to be applied Related OPEN Techniques include State modelling, Service identification and Responsibility identification

5.2.3 For specification of the functional model. For specifying the Functional Model, ONME introduces pre/postcondition specification for every class service, to state what change of state will occur in an object when one of its services is activated, and depending on any potential condition in the current state
In OPEN, there are a number of pre-existing Techniques used to support the creation of the functional model, including standard OO techniques such as Scenario development and Hierarchical task analysis These are readily mapped to (and can therefore be used to instantiate) ONME Techniques as follows The pre/postcondition specification requires two main techniques: the analysis, for every service of what condition must hold in order to have a valid service occurrence (OPEN Technique: Contract specification), and the identification of the effect of every class service, which characterizes the change of state associated with the service execution (OPEN Technique: State modelling)

5.2.4 For specification of the presentation model. For specifying the Presentation Model a predefined set of User Interaction Patterns is used to determine 1) User Interface (UI) properties associated with the scope of the service occurrences (characteristics of the arguments), dependencies between service arguments, offered actions, navigation options); 2) class population queries to select a subset of an object according to a given condition; and 3) specification of presentation styles for the local object state (tabular, register etc). Through the corresponding textual specification of every interaction pattern, the functional system specification is complemented by the characteristics that will guide the interaction between the user and the application More detailed information about the Presentation Model can be found in [27]
In OPEN, there are a number of pre-existing Techniques used to support the creation of the presentation model These are readily mapped to (and can therefore be used to instantiate) ONME Techniques as follows. For determining UI Patterns in the class service scope, the required main technique is the analysis from a usability perspective of what features are required by any service agent to properly interact with the system for executing the service This needs to be done in terms of the particular characteristics of the provided UI patterns. Here, the OPEN Technique of Pattern recognition must be applied in the specific context of UI design. Other OPEN Techniques useful for the presentation model include Dialogue design in UI and Screen painting Subsequent to the original OPEN descriptions, Tasks and Techniques from Usage-Centered Design (UCD) [5] have been added to OPEN [8]. Of relevance to ONME are the new OPEN repository Techniques of Content Modelling (particularly using Canonical abstract components) and Navigation modelling, derived from UCD

5.2 ONME work products

As the OO-Method expressiveness is built upon the four tasks described in Section 5 1, these tasks provide the basis on which to specify the work products that are associated with the main ONME Conceptual Modelling Construction step They are presented in Table 1

Table 1 Tasks and associated Work Products in ONME

TASK	WORK PRODUCT
1 Specification of the Class Model	UML Class Diagram
2 Specification of the Dynamic Model	UML State Diagram / Collaboration Diagram
3 Specification of the Functional Model	Dynamic Logic Axioms including pre and postconditions for class services
4. Specification of the Presentation Model	User Interaction Patterns

The UML Class Diagram required to specify the Class Model is just based on a precise subset of conceptual primitives, i.e. those provided by the OASIS formal framework This makes the use of the UML Class Diagram easier and much clearer According to the techniques previously introduced, classes (including attributes, services and integrity constraints) and relationships between classes (those of association, aggregation and generalization) are specified within this diagram

The UML State Diagram defines for every class the valid life cycles (valid sequence of service occurrences) for the objects from a given class Every transition represents a potential valid service occurrence that can be labelled with the corresponding service precondition Additionally, global transactions (involving services of more than one class) and trigger relationships (to specify automatic activation of services when a given condition holds) are specified through a UML Collaboration Diagram.

To specify class service functionality (the Functional Model), Dynamic Logic Axioms of the form

$f\,[s()]\,f'$

are defined, where f and f' are well-formed formulae built over an alphabet of class attributes and s is the corresponding service whose functionality is being specified The informal meaning of such formulae is that "assuming that f is true, f' will express the new resulting object state after the occurrence of s". These dynamic logic axioms are the result of applying the pre/post specification technique discussed previously

Finally, to specify user interaction properties, the Presentation Model creates a UI specification built from a set of UI patterns belonging to a predefined catalogue of patterns They collect the different situations that have to be considered for UI specification purposes (for instance, selection criteria for looking for determined class instances, display set to fix which attributes should be shown to give more information about a selected object, available action set, to determine what actions can be executed within the scope of a given class service etc).

Summarizing, the use of UML Class Diagrams, UML State Diagrams and UML Collaboration Diagrams as work units maps directly to pre-existing OPEN Work Products, which themselves frequently use UML notations directly [9] while the use of Dynamic Logic Axioms (as a result of the Functional Model) and the use of User Interface Patterns (as a result of the Presentation Model) are inadequately supported by OPEN. Consequently, from this current analytical study, we add two new Work Products to the suite of method fragments in the OPEN repository of method fragments (Figure 1) These are:

1) Dynamic Logic Axioms These can be seen as an advanced, formal representation of pre/post specifications for class services;

2) User Interface Patterns These can be seen as a particular use in a UI context of the well-known patterns-based specifications, widely used in other software engineering environments and described in OPEN's Technique of Pattern recognition

These two new OPEN Work Products are described formally as follows:
NAME Dynamic logic axioms
OPF CLASSIFICATION Dynamic behaviour diagrams
RELATIONSHIP TO EXISTING WORK PRODUCT Adjunct to contract specifications
BRIEF DESCRIPTION A set of well-formed formulae that document pre and postconditions formally in a form appropriate to code generation These are given as
$f\,[s()]\,f'$
where f and f' are well-formed formulae built over an alphabet of class attributes s is the corresponding service whose functionality is being specified
The informal meaning of such formulae is that "assuming that f is true, f' will express the new resulting object state after the occurrence of s".

NAME User interface patterns
OPF CLASSIFICATION Static structure diagrams, UI diagrams
RELATIONSHIP TO EXISTING WORK PRODUCT Application of design patterns in the UI domain
BRIEF DESCRIPTION UI patterns describe various repetitive configurations (patterns) that occur in the UI context These include, for instance, selection criteria for looking for determined class instances, ordering criteria, strategies for grouping service arguments in the context of a service execution, dependency rules among service arguments, a display set to fix the attributes of which should be shown more information about a selected object, available action sets for determining which actions should be executed within the scope of given class service etc.

6. Discussion and Conclusions

ONME is an environment for automated software engineering that is focussed on the modelling activity. It is an implementation of OO-Method, based on OASIS, a formal, object-oriented specification language It provides an operational implementation of MDA, where a PIM to PSM transformation is precisely defined through the definition of the model compiler It takes an OO Conceptual Schema as input (the PIM model) and obtains its corresponding software representation in widely-accepted, three-tier based software architecture, where presentation, functional and data components are properly connected (the PSM model) However, ONME needs to have a precise software process support, which is especially required due to the different type of modellers that are anticipated as using the tool to create the required conceptual schemata. To fix precisely the activities, techniques and work products associated with ONME, the OPEN methodological approach has been selected It offers full lifecycle support, which makes it especially appropriate in this context It does so by utilizing the concepts and tools of method engineering. OPEN consists of a metamodel underpinning a generated set of method fragments, stored in a repository (Figure 1). The user can then select from these pre-defined method fragments and construct their own methodology to exactly fit their local conditions
Here we have used this ME approach, based on OPEN, to re-create the modelling elements of the ONME approach In this way, the required software process support is given. This is a needed, prior step towards the full implementation of an ONME-specific full lifecycle methodology – which will include full support for project management, usability, team and

individual skills and characteristics as well as the support of a fully iterative and incremental lifecycle model

Incidentally, as we created an environment from which to instantiate ONME, we identified two work products recommended by OO-Method not previously described in the OPEN literature. These two work products, Dynamic logic axioms and User interface patterns, have consequently been recommended here for formal adoption as method fragments in the OPEN repository.

As ONME is being enriched with the addition of a Requirements Analysis Process (RAP) [12], further work will be accomplished to enrich the current, OPEN-based proposal, in order to include the quoted RAP extension in the software process associated with ONME

References

1. Beck, K. and Cunningham, W., 1989, "A laboratory for teaching object-oriented thinking", Procs. 1989 OOPSLA Conference. ACM SIGPLAN Notices, 24(10), 1-6

2. Brinkkemper S., Saeki M. and Harmsen F., 1998, "Assembly techniques for method engineering", Procs. CAISE 1998, Springer Verlag, Berlin, Germany, pp. 381-400

3. Brinkkemper, S., 1996, "Method engineering: engineering of information systems development methods and tools", Inf. Software Technol., 38(4), pp. 275-280

4. Cockburn A.S., 2000, "Selecting a project's methodology", IEEE Software, 17(4), pp. 64-71

5. Constantine L.L. and Lockwood L.A.D., 1999, Software for Use, Addison-Wesley, Reading, MA, USA.

6. Firesmith D.G. and Henderson-Sellers B., 2002, The OPEN Process Framework. An Introduction, Addison-Wesley, Harlow, Herts, UK.

7. Graham I., Henderson-Sellers B. and Younessi B., 1997, The OPEN Process Specification, Addison-Wesley, Harlow, Herts, UK.

8. Henderson-Sellers B. and Hutchison J., 2003, "Usage-Centered Design (UCD) and the OPEN Process Framework (OPF)", Performance by Design. Procs. forUSE2003, Second International Conference on Usage-Centered Design (ed. L.L. Constantine), Ampersand Press, Rowley, MA, USA, pp. 171-196.

9. Henderson-Sellers B. and Unhelkar B., 2003, OPEN Modeling with UML, Addison Wesley Longman, Ltd., London, UK.

10. Henderson-Sellers B., 2003, "Method engineering for OO system development", Comm. ACM, 46(10), pp. 73-78.

11. Henderson-Sellers B., Simons A.J.H. and Younessi H., 1998, The OPEN Toolbox of Techniques, Addison-Wesley Longman Ltd., London, UK.

12. Insfran E., Pastor O., Wieringa R., 2002, "Requirements engineering-based conceptual modelling", Requirements Engineering Journal, Springer-Verlag, 7(2), pp. 61-72

13. Klooster M., Brinkkemper S., Harmsen F. and Wijers G., 1997, "Intranet facilitated knowledge management: A theory and tool for defining situational methods", Procs. CAISE 1997, Springer Verlag, Berlin, Germany, pp. 303-317

14. Nguyen V.P. and Henderson-Sellers B., 2003, "OPENPC: a tool to automate aspects of method engineering", Procs. ICSSEA 2003. Volume 5, 7pp

15. OMG, 2003, MDA Guide Version 1.0.1, omg/2003-06-01, Object Management Group. http://www.omg.org/cgi-bin/doc?omg/03-06-01, 62pp

16. OMG, 2001, Unified Modeling Language Specification, v1.4. formal/01-09-67. Object Management Group. http://www.omg.org/docs/formal/01-09-67.pdf, 566pp

17. Pastor O., Gomez J., Insfrán E. and Pelechano V., 2001, "The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming", Information Systems, 26, pp. 507-534.

18 Pastor O , Hayes F and Bear S., 1992, "OASIS: An object oriented specification language", Procs CAiSE 1992, LNCS593, Springer-Verlag, Berlin, Germany, pp. 348-363.
19 Pastor O , Molina J.C and Iborra E., 2004, "Automated production of fully functional applications with OlivaNova Model Execution", ERCIM News No.57, April 2004
20 Pelechano V , Pastor V. and Insfrán E , 2002, "Automated code generation of dynamic specializations: an approach based on design patterns and formal techniques", Data & Knowledge Engineering, 40, pp. 315-353
21. Ralyté J and Rolland C , 2001, "An assembly process model for method engineering", Advanced Information Systems Engineering), LNCS2068, Springer-Verlag, Berlin, Germany, pp 67-283
22 Rolland C and Prakash N., 1996, "A proposal for context-specific method engineering", Procs IFIP WG8.1 Conf. on Method Engineering, pp 191-208.
23 Rolland, C., Prakash, N and Benjamen, A., 1999, A multi-model view of process modelling, Requirements Eng. J , 4(4), pp 169-187
24. Saeki M., 2003, "CAME: the first step to automated software engineering", Process Engineering for Object-Oriented and Component-Based Development. Procs OOPSLA 2003 Workshop, Centre for Object Technology Applications and Research, Sydney, Australia, pp. 7-18.
25. Ter Hofstede A.H.M and Verhoef T F , 1997, "On the feasibility of situational method engineering", Information Systems, 22, pp 401-422.
26 Van Slooten K and Hodes B , 1996, "Characterizing IS development projects", Proceedings of the IFIP TC8 Working Conference on Method Engineering: Principles of method construction and tool support (eds S Brinkkemper, K. Lyytinen and R Welke), Chapman & Hall, London, UK, pp. 29-44
27 Molina, P J., Melia, S and, Pastor, O , 2002. "User interface conceptual patterns". Proceedings of the 9th Int Conference on Design, Specification, and Verification of Interactive Systems (DSV-IS'2002), Rostock, Germany, LNCS 2545, Springer-Verlag, pp.159-173